Promoting Cognitive Design Patterns using Software Integrated Development Environments[1]

Vincent Schmidt, vincent.schmidt@wpafb.af.mil; **and Christopher R. Hale**, halec@saic.com

Whether we are designing a new system, or maintaining and upgrading an existing one, our goal must be to ensure the system supports the work to be accomplished. The user will shun systems with poor work support or poor usability, and such a system will ultimately fall into disuse. The costs of systems with poor work support are staggeringly high, both in actual development dollars and total cost of ownership, which includes the cost of humans to operate and maintain systems This article emphasizes the importance of designing systems that adequately account for the work requirements and discusses some tools that can help engineers achieve that goal.

Typically, we think of *work* as referring to a collection of technical factors, bound together within an appropriately structured environment. However, we must recognize the importance of the human as a key component of the system. Human users are frequently producers of inputs, consumers of outputs, and generators of system controls. As such, users are critical components of the system and must be explicitly included in the system design.

INCOSE defines *human systems integration* (HSI) as "the interdisciplinary technical and management process for integrating human considerations within and across all system elements; an essential enabler to systems engineering practice" (Haskins 2007). The HSI part of the systems engineering process certainly includes the physical integration of the human into the system, as well as recognizing the cognitive capabilities and constraints the human user brings to the performance of the system as a whole. We find it feasible to identify, codify, and incorporate this human cognitive component into the system design in a more or less formal manner.

HSI practitioners have evolving models that capture human cognitive requirements. Some models concentrate on describing cognition at an atomic modeling level, which system architects and implementers could use to account for various cognitive system design trade-offs early in the design process. O'Malley et al. (2008) provide a model that integrates cognitive descriptions into the United States Department of Defense Architecture Framework. Another model is the low-level descriptive solution presented by Hale and Schmidt (2008) and by Hale (2006 and 2008), which is a comprehensive mechanism for accurately identifying and documenting cognitive requirements. This solution builds an integrated network of descriptive cognitive elements and maps the resulting network to specific system cognitive requirements in context. Of course, a host of alternative models exists, which we do not have room to discuss in detail; some concentrate on cognitive psychology, while others are oriented strictly toward the more formal mathematics of system design.

System designers commonly use patterns as a descriptive mechanism to communicate high-level design requirements and criteria. Subspecialties within the engineering and design communities have developed their own design philosophies and symbologies in order to foster an accurate communication of the design and design requirements. Consider the popular notion of electronic circuit design, where patterns and symbols shown in schematic diagrams are used effectively to communicate the expectation of specific implementation details. Similarly, the software engineering community frequently relies on the use of the Unified Modeling Language to capture the structure, software representation, and data exchange expectations within integrated software systems. Even in these examples, we develop and use higher-level patterns to improve communications and reduce descriptive complexity. More design pattern concepts are described by Conrad and Stanard (2007), Stanard et al. (2006), and Stanard and Wampler (2005).

Table 1. Cognitive work element for operational assessment

Acquire	Communicate	Compare	Infer
Decide	Discriminate	Estimate	Integrate
Assign	Aggregate	Evaluate	Identify
Choose	Describe	Generate	Interpret
Classify	Detect	Match	Plan
Monitor	Recognize	Prioritize	Verify

The enthusiasm of the computing community has fueled a resurgence of interest in the concept of "design patterns;" and the use of patterns is certainly not limited to areas strictly considered as software engineering and development tasks. We find that we can effectively generate a series of design patterns to capture the essence of cognitive processes and tasks. These new design patterns are an invaluable asset for formalizing the system architecture and helping to ensure that the implementer has met the documented cognitive system requirements.

We can also apply design patterns to software systems. We recently presented preliminary concepts for cognitive design patterns (Hale and Schmidt 2008). The basis of our work was the introduction of a small number of reusable and general system-level cognitive work elements (CWEs) as building blocks of cognition. We can combine these CWEs to fully describe and encapsulate system cognitive requirements. A system using the formal CWE descriptions within the implementation shows full traceability back to the original cognitive requirements. Table 1 shows a set of cognitive work elements for an operational assessment task. The field of cognitive science defines these terms with very specific meanings. They are typical of CWE terminology at a microcognitive level of analysis.

Here, we are extending the ideas of our previous work by describing the practical implementation goals of a modular

» *continues on next page*

1. Approved for public release by AFMC / 711 HPW, # 88ABW-2008-1130.

Schmidt et al. *continued*

software development solution using the CWE concept.

Integrated Development Environments

Professional programmers rely heavily on integrated development environments (IDEs) for both rapid prototyping and production system design. IDEs typically include a full suite of development tools, including outstanding programming language and library documentation, excellent software formatting, integrated revision control, and superb compiler and debugging facilities. Many integrated development environments are extremely complex and might even have support for multiple simultaneous programming languages, as well as for multiple hardware and operating system platforms. An additional valuable feature of several IDEs is the integration of a graphical user interface design subsystem. This allows the programmer to graphically build screen layouts using icons, widgets, wizards, and graphical user interface design patterns.

Software system developers often prefer to use integrated development environments because of the benefits they offer, including real-time type checking capabilities and related constraint testing that occurs while coding, thus reducing the rate at which bugs are introduced into code. In addition, library and module control mechanisms increase the capacity for code reusability. Pattern libraries are also beneficial, offering debugged code at an appropriate level of granularity, and improving the ability of the designer and implementer to communicate the expectations of documented requirements. All of these items contribute to a sense of completeness offered by the IDE.

Many popular integrated development environments are extensible, allowing third party plug-in modules to introduce additional functionality. Many modules introducing advanced graphical user interface widget components already exist for several IDEs. Other modules might bring programming language enhancements, a graphical user interface, or additional widgets to the IDE. Plug-in modules are often the ideal mechanism for incorporating new design pattern libraries, such as a pattern for a cognitive

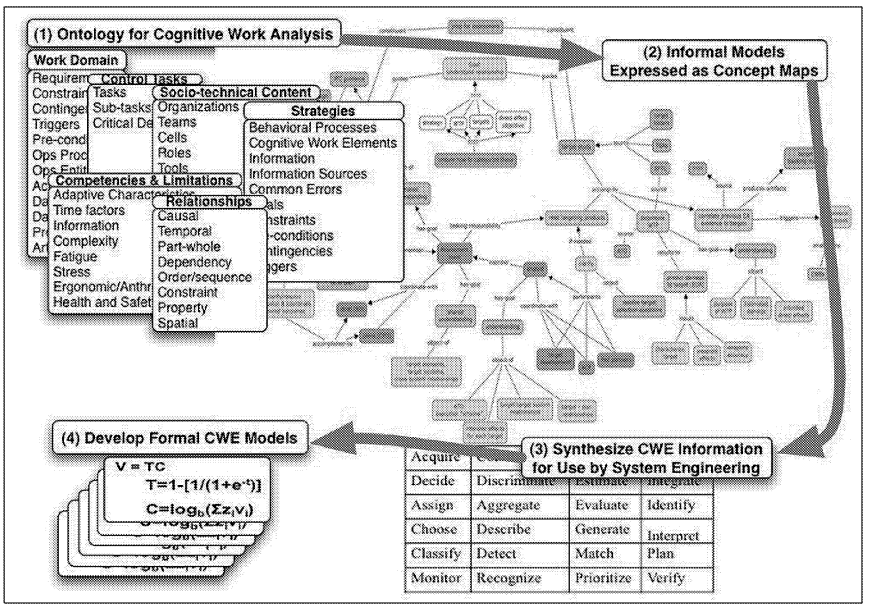

Figure 1. Development cycle for cognitive work elements

work element. Since developers can share modules, a specialized design pattern module could easily be distributed to others wishing to take advantage of the same functionality.

A cognitive design pattern plug-in module would augment the integrated development environment with a selection of cognitive patterns, along with documentation describing the patterns and their use. This completely self-contained module would introduce the concept of the cognitive component of human systems integration, and should assist the programmer by promoting the inclusion of cognitive requirements into the system implementation at an appropriate level of detail. Merely making such a module available will also increase the popularity and utility of the oft-ignored cognitive aspect of the HSI spectrum.

Using Cognitive Design Patterns

We incorporate cognitive design patterns into the integrated development environment to promote actualizable (if abstract) solutions to real, documented cognitive requirements. When a cognitive work element is chosen from a palette of available work elements, a wizard would guide the programmer through a selection of relevant visualizations promoting the element's implementation. By suggesting one or more visual representations of the CWE, the programmer will reduce the risk of misrepresenting the cognitive requirement.

When we use cognitive design patterns from the palette, we also make requirements traceable. If we specify the requirements in terms of CWEs, then the cognitive design patterns implementing the CWEs have essentially a 1:1 relationship with the set of CWEs. The developer can point to the use of specific cognitive design pattern components as partial justification for meeting the documented cognitive requirements.

Note that a collection of cognitive design patterns is a development tool; merely making this tool available for a software developer certainly does not ensure that the resulting software is a reasonable HSI design. Similarly, given a good HSI design, the developer can still misuse the concepts of the cognitive design pattern and misrepresent the intended design. However, having a well-developed library of cognitive design pattern concepts does help build a hedge against bad designs created out of ignorance.

Building Cognitive Design Patterns

The availability of libraries of cognitive design patterns within integrated development environments is an asset for software developers. The benefits of access to vetted and precoded cognitive concepts—no more coding from

System Requirement	Acquire	Assign	Aggregate	Choose	Classify	Communicate	Compare	Decide	Describe	Detect	Discriminate	Estimate	Evaluate	Generate	Identify	Infer	Integrate	Interpret	Match	Monitor	Prioritize	Plan	Recognize	Verify
The system shall aid in determining if operational objectives will be achieved	✓				✓							✓				✓		✓					✓	
The system shall aid in determining if actual events coincide with the plan															✓					✓				✓
The system shall aid in and allow the assessment of the feasibility of the corrective actions							✓					✓	✓		✓	✓								✓
The system shall aid in correlating combat assessment results with tactical tasks, tactical objectives, and operational objectives					✓								✓				✓	✓						
The system shall allow and aid in the balancing of risk vs. benefits					✓							✓			✓	✓					✓			

Figure 2. Mapping functional requirements to cognitive work element patterns

scratch each time—is guaranteed to improve the quality and reliability of the resulting software system. However, the library must be developed before it can be included as a development module, so we now address what must be done to build a cognitive design pattern for inclusion in the cognitive design pattern library. We must first gather the list of cognitive work elements to be included in the library. Each CWE has a very specific definition, and over time, the list of CWEs should become complete. (In fact, we believe our list to be comprehensive, but our claim has not yet been rigorously tested.)

Our list of CWEs is originally derived from concept maps describing cognitive work requirements. These concept maps are often best understood and constructed by psychologists, human factors specialists, and trained requirements analysts, although it might also be valuable for technologists to participate in developing these products. The concept maps not only provide a mechanism for capturing work requirements; we can also glean relevant cognitive structures from analyzing the concept maps. These cognitive structures consist of one or more related units of cognition (cognitive work elements). Figure 1 shows an abstraction of the cycle of operations producing the collection of CWEs (from table 1) for an operational assessment work task.

Once the list of related CWEs is defined, we must determine how to model the CWEs as a pattern. We should represent the models themselves in terminology appropriate for systems engineering work, since a significant goal of using CWEs is to capture and document cognitive requirements.

We have a variety of existing models for integrating human systems integration requirements and cognitive requirements into the systems engineering design documentation. One example includes integration of cognitive requirements into a Department of Defense Architecture Framework representation (O'Malley et al. 2008). Developing a pattern representation for the CWE is a slightly different issue, however. The most useful patterns are those that could be introduced as advanced graphical user interface components, as a library module for an existing integrated development environment. We would need to abstract each cognitive work element as a pattern, complete with one or more representative graphical visualizations and algorithms. These representations would serve as suggestions for the technologist implementing the requirement, resulting in an approach for formally specifying the CWE models into the software architecture. One way to implement CWE operations is to use a dynamic software wizard to guide the developer through the CWE options, potentially showing a sample visual graphic based on the selections made through the wizard.

Once we complete the wizard for the CWE element pattern, then we would automatically inject operational code or code templates into the code, along with supportive comments and a reference (provided by the programmer) to the cognitive requirement or requirements met by this CWE. This would make the CWE widgets helpful as both a communication and prototyping tool. We would include requirement references in the instantiated code to produce requirements traceability, which is very important. At any time, we could review the code to find an exact correspondence to specific requirements. This would provide the technologist a mechanical way to easily account for all of the requirements, while also yielding a justification for how (or why) particular implementations, visualizations, and algorithms are used within the system. Figure 2 depicts an example of mapping functional requirements onto CWE patterns.

The complete implementation of CWEs as modular cognitive design patterns helps the programmer to determine what requirement is met, what is an appropriate visualization or algorithm to implement this CWE, and what parts of the implementation must be static within the software, versus what parts can be dynamically modified by the end user.

Future Work

Many integrated development environments already provide the facility to add advanced third-party plug-ins and modular extensions. Our contribution to the state of the technology currently includes the development of cognitive work element and cognitive design pattern theory. We expect to use the basic structure outlined in this article to begin implementing these ideas for use with one or more popular integrated development environments, with the input of graphical user interface builders, as the next phase of this research. There are clear advantages to including explicit, reusable cognitive design patterns (implementations of cognitive work elements) directly into a software system's source code, with traceability back to

» *continues on next page*

Three Principles for Effectively Addressing Cognition

Sterling Wiggins, swiggins@ara.com

Principle 1: The performance of a system is the result of combined human and technological performance.

Mission performance is preeminent, not human or technological performance: both are necessary when building sociotechnical systems, but neither is sufficient. The goal is to combine human and technological capabilities to build systems that transcend the limitations of the individual human or technology.

Principle 2: Cognitive aspects of work must be addressed if the right system is to be built right.

Cognitive systems engineers do not need to participate in the development process to achieve good system design, but flawless systems engineering without consideration for cognitive aspects of the system can result in building the wrong system, right. Cognitive systems engineering is a structured approach for ensuring that cognitive aspects are addressed and that the right system is built right. It serves to identify and describe system features that are linked to the cognitive work that needs to be supported.

Principle 3: Cultivate a culture of productive and informed compromise.

Compromises in human and technical areas will be necessary to build a system on time, on schedule, and that is acceptable to its intended operators. Program managers and lead systems engineers need to populate the project environment with the attitudes, tools, and opportunities required for productive compromise. For example, cognitive systems engineers should sit on the systems engineering and other integrated product teams so that there will be a continual dialogue.

Bibliography of Cognitive Engineering Resources

Burns, C. M., and J. R. Hajdukiewicz. *Ecological Interface Design.* Boca Raton, FL: CRC Press, 2004.

Cook, N., and F. Durso. *Stories of Modern Technology Failures and Cognitive Engineering Successes.* Boca Raton, FL: CRC Press, 2008.

Crandall, B., G. Klein, and R. R. Hoffman. *Working Minds: A Practitioner's Guide to Cognitive Task Analysis.* Cambridge: MIT Press, 2006.

Deal Corp., Web site for the Acquisition Practitioner Support Environment. http://acprac.com.

Klein, G. *Sources of Power: How People Make Decisions.* Cambridge, MA: MIT Press, 1998.

MITRE Corporation. *A Survey of Cognitive Engineering Methods and Uses.* http://mentalmodels.mitre.org/cog_eng/.

Norman, D. A. *The Design of Everyday Things.* New York: Doubleday, 1990.

Rasmussen, J., A. M. Petjersen, and L. P. Goodstein. *Cognitive Systems Engineering.* New York: Wiley, 1994.

SA Technologies, *Professional Publications.* http://www.satechnologies.com/publications. (Situational awareness approaches.)

TRACE-SE: The Resource for Applied Cognitive Engineering and Systems Engineering. https://www.trace-se.com.

Usernomics. *User Interface Design.* http://www.usernomics.com/user-interface-design.html.

Vicente, K. J. *Cognitive Work Analysis.* Mahwah, NJ: Lawrence Erlbaum, 1999.

Schmidt et al. *continued*

the cognitive requirements. The fielded software system will be easier to develop, maintain, and use, with a demonstrated capacity for meeting the user's system expectations and supporting the work requirements.

References

Conrad, K., and T. Stanard. 2007. Advanced design patterns: Enabling designers of complex systems. *Proceedings of the 2007 International Conference on Software Engineering Research and Practice* (Las Vegas, NV): 1—7.

Hale, C. R. 2006. Executable requirements for visualization design. Paper presented at the 2006 Human Factors and Ergonomics Society Conference (San Francisco, CA).

_____. 2008. Visualization design using an integrated joint cognitive system development methodology. Proceedings of the 2008 Human Factors and Ergonomics Society Conference.

Hale, C. R., and V. Schmidt. 2008. Cognitive design patterns. *Paper presented at of the 2008 International Conference on Software Engineering, Research, and Practice* (Las Vegas, NV): 611—616.

_____. 2008. Four challenges, and a proposed solution, for cognitive system engineering–system development integration. Paper presented at the 2008 Industrial Engineering Research Conference (Vancouver, BC).

Haskins, C., ed. 2007. *Systems engineering handbook: A guide for system life cycle processes and activities.* Version 3.1. Rev. by K. Forsberg and M. Krueger. Seattle: INCOSE.

O'Malley, D., Zall, J., Colombi, J., and Carl, J. 2008. Integrating cognition into system design. *Proceedings of the 2008 International Conference on Software Engineering, Research, and Practice* (Las Vegas, NV): 604–610.

Stanard, T., and J. Wampler. 2005. Work-centered HCI design patterns. Paper presented at the conference, *INTERACT 2005: Communicating Naturally through Computers* (Rome, Italy).

Stanard, T., Osga, G., Wampler, J., and Conrad, K. 2006. HCI design patterns for C2: A vision for a DoD design reference library. Paper presented at the 2006 Command & Control Research and Technology Symposium (San Diego, CA).